HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200209059-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Jay D. Knitter

Confirmation No.: 9770

Application No.: 10/635,815

Examiner: Christopher D. Biagini

Filing Date: August 7, 2003

Group Art Unit: 2142

Title: METHOD AND APPARATUS FOR IDENTIFYING A MESSAGE SOURCE IN A NETWORK

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on January 24, 2008.

☒ The fee for filing this Appeal Brief is $510.00 (37 CFR 41.20).

☐ No Additional Fee Required.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

☐ (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d)) for the total number of months checked below:

☐ 1st Month $120	☐ 2nd Month $460	☐ 3rd Month $1050	☐ 4th Month $1640

☐ The extension fee has already been filed in this application.

☒ (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account 08-2025 the sum of $ 510. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees.

☐ A duplicate copy of this transmittal letter is enclosed.

☐ I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:
Commissioner for Patents, Alexandria, VA 22313-1450
Date of Deposit:

OR

☐ I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571)273-8300.
Date of facsimile:
Typed Name:
Signature:_____

Respectfully submitted,
Jay D. Knitter
By /William T. Ellis/ Reg. No. 59,396
William T. Ellis
Attorney/Agent for Applicant(s)
Reg No. : 26,874
Date : March 24, 2008
Telephone : (202) 672-5485

Atty. Dkt. No. 200209059-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Appellant:	Jay D. Knitter
Title:	METHOD AND APPARATUS FOR IDENTIFYING A MESSAGE SOURCE IN A NETWORK
Appl. No.:	10/635,815
Filing Date:	08/07/2003
Examiner:	Christopher D. Biagini
Art Unit:	2142
Confirmation Number:	9770

BRIEF ON APPEAL

Mail Stop Appeal Brief - Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Under the provisions of 37 C.F.R. § 41.37, this Appeal Brief is being filed. Please charge $510.00 to cover the 37 C.F.R. 41.20(b)(2) appeal fee to Deposit Account 08-2025.

REAL PARTY IN INTEREST

The real party in interest is Hewlett-Packard Company.

RELATED APPEALS AND INTERFERENCES

Application No. 10/448646 incorporated by reference in the present application was filed on May 30, 2003. A Notice of Appeal was filed on July 23, 2007. An appeal brief was filed on September 20, 2007. The application has been forwarded to the Board of Patent Appeals and Interferences for a decision on the appeal.

STATUS OF CLAIMS

The present appeal is directed to claims 1-22 which are the claims under consideration. A copy of pending claims 1-29 is attached herein in the Claims Appendix.

Claims 1-22 were rejected under 35 U.S.C. § 112, first paragraph, as failing to comply with the enablement requirement.

Claims 6-9, 15-18 and 21 were rejected under 35 U.S.C. § 112, second paragraph, as being indefinite.

STATUS OF AMENDMENTS

Claims 1-22 were initially pending in the application filed on August 7, 2003.

Claims 1, 11, 19 and 22 were amended in an Amendment and Reply filed July 16, 2007.

Claims 1, 11, 19 and 22 were amended in an Amendment and Reply filed November 26, 2007. However, in an Advisory Action mailed December 19, 2007, the Examiner indicated that the amendments of November 26th would not be entered for the purposes of appeal.

This Appeal Brief is being filed within the statutory two month period after the filing of the Notice of appeal on January 24, 2008.

SUMMARY OF CLAIMED SUBJECT MATTER

Claims 1, 11, 19 and 22 are independent claims.

Independent claim 1 is directed to a method of identifying a message source in a network. Figure 2, step 220 and page 3, line 29 – page 4, line 5 of the application describe receiving a method call from a client computer to invoke an object on a server. Figure 2, step 230 and page 4, lines 5-9 of the application describe packaging the method call in a message to be sent from a client server to the data server via the network. Figure 2, step 240 and page 4, lines 25-28 of the application disclose identifying, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on

the data server. Figure 2, step 250 and page 4, line 28 – page 5, line 2 of the application describe transmitting the message to the data server.

Independent claim 11 is directed to a client server configured to transmit messages to a data server via a network. Figure 1, page 3, lines 17-23, Figure 2, step 220 and page 3, line 29 – page 4, line 5 of the application describe a client computer interface configured to receive a method call from a client computer to invoke an object on the data server. Figure 2, step 230 and page 4, lines 5-9 describe a data processing unit coupled to the client computer configured to package the method call in a message to be sent from a client server to the data server via the network. Figure 2, step 240 and page 4, lines 25-28 of the application disclose a data processing unit coupled to the client computer configured to identify, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on the data server. Figure 2, step 250 and page 4, line 28 – page 5, line 2 of the application describe a data processing unit coupled to the client computer configured to transmit the message to the data server.

Claim 19 is directed to a program product, embodied in a computer readable medium, comprising machine-readable program code for causing, when executed, a computer to graphically emulate a network including at least a client computer, a client server and a data server. Figure 2, step 220 and page 3, line 29 – page 4, line 5 of the application describe the program product performing the method step of receiving a method call from a client computer to invoke an object on a server. Figure 2, step 230 and page 4, lines 5-9 of the application describe the program product performing the method step of packaging the method call in a message to be sent from a client server to the data server via the network. Figure 2, step 240 and page 4, lines 25-28 of the application disclose the program product performing the method step of identifying, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on the data server. Figure 2, step 250 and page 4, line 28 – page 5, line 2 of the application describe the program product performing the method step of transmitting the message to the data server.

Claim 22 is directed to an apparatus configured to identify a message source in a network. Figure 2, step 220 and page 3, line 29 – page 4, line 5 of the application describe a

means for receiving a method call from a client computer to invoke an object on a server. Figure 2, step 230 and page 4, lines 5-9 of the application describe a means for packaging the method call in a message to be sent from a client server to the data server via the network. Figure 2, step 240 and page 4, lines 25-28 of the application disclose a means for identifying, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on the data server. Figure 2, step 250 and page 4, line 28 – page 5, line 2 of the application describe a means for transmitting the message to the data server.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Accordingly, the issue on appeal is whether the examiner erred in:

Finally rejecting claims 1-22 under 35 U.S.C. § 112, first paragraph, as failing to comply with the enablement requirement.

Finally rejecting claims 6-9, 15-18 and 21 under 35 U.S.C. § 112, second paragraph, as being indefinite.

ARGUMENT

Rejection of Claims 1-122 under 35 U.S.C. § 112, First Paragraph

As a preliminary matter, Appellant notes that the level of predictability in the computer arts is generally considered predictable. The amount of guidance or direction needed to enable the invention is inversely related to the amount of knowledge in the state of the art as well as the predictability in the art. *In re Fisher*, 427 F.2d 833, 839, 166 USPQ 18, 24 (CCPA 1970). The more that is known in the prior art about the nature of the invention, how to make, and how to use the invention, and the more predictable the art is, the less information needs to be explicitly stated in the specification. *See* MPEP § 2164.03. Here, in contrast to the above teachings, the Examiner, while acknowledging the level of predictability in the art is high, requires that the specification provide significant guidance and direction.

As long as the specification discloses at least one method for making and using the claimed invention that bears a reasonable correlation to the entire scope of the claim, then the

enablement requirement of 35 U.S.C. § 112 is satisfied. *In re Fisher*, 427 F.2d 833, 839, 166 USPQ 18, 24 (CCPA 1970). Paragraphs [0015]-[0022] and Figs. 2-5 of the present application disclose a method for carrying out the claimed invention. Further, in view of the fact that significant guidance in the art is not required and Application Nos. 10/448,646 and 10/449,555 are incorporated by reference, Appellant respectfully traverses the rejection and requests that the rejection be withdrawn.

In the Office Action dated September 26, 2007, the Examiner sets forth several factors in support of the assertion that claims 1-22 fail to comply with the enablement requirement. In the Office Action, the Examiner argues that the "features that Appellant argues distinguish the claims from the prior art are those that are not adequately described." Specifically, the Examiner asserts that the Appellant argued that the distinguishing feature of the claimed invention is the use of a comparison algorithm on a client server to identify an object on the client computer that is invoking the object on [a] data server but the specification does not adequately describe how this operation occurs. Appellant respectfully disagrees.

Claims 1-22 are enabled by the specification, which includes U.S. Patent Application Nos. 10/448,646 and 10/449,555 which were incorporated by reference on page 1 of the specification. The claimed method recites that an object on a client computer is identified from an execution stack using a comparison algorithm. On page 6 of the September 26th Office Action, the Examiner argues that the specification does not adequately describe how a comparison algorithm on a client server is used to identify an object on the client computer that is invoking the object on a data server. However, paragraph [0019] of the specification clearly sets forth this operation. Paragraph [0019] states "[t]he identifier (fully qualified class name) of the source of the SOAP call is stored in a SOAP header which is part of the message transmitted from the client computer 11 to the data server 140 (via client server 120)." The identifier indicates the object on the client computer that is invoking the object on the data server. As disclosed in paragraph [0019] the identifier is transmitted in the header of the method call that is transmitted from the client computer to a data server via the client server. The identifier is a portion of the execution stack and identifies the object name that invoked the method to send the message to the data server. *See* page 7, lines 27-29 ("However, the entire stack is not in the header, only the class name that invoked the method to send the

message to the data server 140."). Since a portion of the execution stack of the client computer, namely an identifier indicating the object on the client computer that is invoking the object on the data server is included in the header of the method call sent from the client computer to the client server, the client server is able to examine the header of the method call to identify the object that is invoking the object on the data server. *See* page 7, lines 29-30 ("The identifier can the [sic] be retrieved by an administrator by simply examining the header of the message.").

Paragraph [0016] specifically recites an algorithm for implementing the claimed invention. The code in paragraph [0016] is for illustration purposes only. The client class generates a method call (sendSOAPMessage()) to invoke an object on the data server. The method sendSOAPMessage() creates a new class Client3 which invokes a method called findSourceOfSOAPMessage(). The findSourceOfSOAPMessage() method uses an algorithm (shown on page 6) to find and return the source of the SOAP message (object that generates the method call). This code is executed on the client computer. The source of the SOAP message is identified in a SOAP header which is apart of the method transmitted to the data server via client server. As stated above, since a portion of the execution stack of the client computer, namely an identifier indicating the object on the client computer that is invoking the object on the data server is included in the header of the method call sent from the client computer to the client server, the client server is able to examine the header of the method.

In the Final Office Action, the Examiner states that the client executes on the same machine as the comparison algorithm and would not be functional in the claimed embodiment. However, Appellant notes that a "client server" need not be a separate machine. The client server can be a program that runs on the client computer. *See* Corba Glossary (http://www.ooportal.com/corba-fundamentals/glossary.html), last viewed March 10, 2008. Accordingly, the claims, given their most reasonably broad interpretation, would cover a system where the client server is a program running on the client computer. Further, one skilled in the art, e.g., Java programming, given the algorithm in paragraph [0016] would be able to implement the algorithm in a client server environment (where the client and server are separate machines) using any combination of Java technologies including, for example, Java Server Pages, Java Servlets, Java Enterprise Beans and similar technologies. Moreover,

in view the fact that less information needs to be explicitly stated in the specification because the level of predictability is high, the totality of evidence suggests that it would not require undue experimentation to make and use the claimed invention.

Rejection of Claims 6-9, 15-18 and 21 under 35 U.S.C. § 112, Second Paragraph

Claims 6-9, 15-18, and 21 were rejected as being indefinite for failing to point out which Simple Object Access Protocol (SOAP) the claims are directed to. As stated in the "Background of the Invention," SOAP is a "protocol layered on top of HTTP ... which allows Automation objects to be invoked over the Internet via Web servers." Further, a precise and clear definition of the SOAP protocol can be obtained from paragraph [0004] of the specification which describes briefly how the SOAP protocol operates.

Given the definition set forth in the specification and the clear context given in paragraph [0004], the term SOAP has a clear meaning which is sufficiently precise and definite.

The protocol described in U.S. Patent No. 6,457,066 ("the '066 patent") and the protocol implemented by Apache Axis were included as examples of implementations of SOAP. In other words, these implementations are only subsets of a larger SOAP class that is compatible with the applicant's invention. Thus claims 6-9, 15-18, and 21 do not refer only to the '066 patent's SOAP implementation or to the Apache Axis SOAP implementation, but the claims refer to any implementation that a person having ordinary skill in the art would recognize as a SOAP implementation. The wide ranging application of the applicant's invention is buttressed by its compatibility with "other software and protocols, such as various TCP messaging protocols." ([0011]).

Accordingly, Applicant respectfully requests that the rejection of claims 6-9, 15-18 and 21 be withdrawn.

CONCLUSION

In view of above, appellants respectfully solicit the Honorable Board of Patent Appeals and Interferences to reverse the rejections of claims 1-22 under 35 U.S.C. § 112 and pass this application on to allowance.

Respectfully submitted,

Date 3/24/08

By Walter K. Robinson Reg. No. 59,396

HEWLETT-PACKARD COMPANY
Customer Number: 22879
Telephone: (202) 672-5485
Facsimile: (202) 672-5399

William T. Ellis
Attorney for Appellant
Registration No. 26,874

Walter K. Robinson
Attorney for Appellant
Registration No. 59,396

CLAIMS APPENDIX

1. (Previously Presented) A method of identifying a message source in a network, comprising:

receiving a method call from a client computer to invoke an object on a data server;

packaging the method call in a message to be sent from a client server to the data server via the network;

on the client server, identifying, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on the data server; and

transmitting the message to the data server.

2. (Original) The method of claim 1, further comprising:

on the client computer, generating the method call to invoke the object on the data server.

3. (Original) The method of claim 2, wherein transmitting the message to the data server transmits an identifier of an object on the client computer invoking the object on the data server along with the message.

4. (Original) The method of claim 3, wherein the identifier is stored in a header of the message.

5. (Original) The method of claim 3, wherein the identifier comprises a fully qualified class name.

6. (Original) The method of claim 1, wherein the message comprises a simple object access protocol (SOAP) message.

7. (Original) The method of claim 6, wherein packaging the method call in a message comprises building up a SOAP request.

8. (Original) The method of claim 7, wherein transmitting the message comprises implementing a SOAP application programming interface (API).

9. (Original) The method of claim 8, wherein the SOAP API comprises a messaging API.

10. (Original) The method of claim 2, further comprising:

displaying a Web service graphical component representing the object; and

displaying an interconnecting graphical component representing an associated interaction between the client computer and the data server.

11. (Previously Presented) A client server configured to transmit messages to a data server via a network, comprising:

a client computer interface configured to receive a method call from a client computer to invoke an object on the data server; and

a data processing unit coupled to the client computer interface, the data processing unit being configured to:

package the method call in a message to be sent from the client server to the data server via the network;

identify, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on the data server; and

transmit the message to the data server.

12. (Original) The client server of claim 11, wherein the message is transmitted along with an identifier of an object on the client computer invoking the object on the data server.

13. (Original) The client server of claim 12, wherein the identifier is stored in a header of the message.

14. (Original) The client server of claim 12, wherein the identifier comprises a fully qualified class name.

15. (Original) The client server of claim 11, wherein the message comprises a simple object access protocol (SOAP) message.

16. (Original) The client server of claim 15, wherein packaging the method call in a message comprises building up a SOAP request.

17. (Original) The client server of claim 16, wherein transmitting the message comprises implementing a SOAP application programming interface (API).

18. (Original) The client server of claim 17, wherein the SOAP API comprises a messaging API.

19. (Previously Presented) A program product, embodied in a computer readable medium, comprising machine-readable program code for causing, when executed, a computer to graphically emulate a network including at least a client computer, a client server, and a data server, the program product graphically emulating the network performing method steps of:

on the client computer, generating a method call to invoke an object on the data server;

packaging the method call in a message to be sent from the client server to the data server via the network;

on the client server, identifying an identifier of an object on the client computer invoking the object on the data server from an execution stack through a comparison algorithm; and

transmitting the message to the data server.

20. (Original) The program product of claim 19, wherein the identifier comprises a fully qualified class name.

21. (Original) The program product of claim 19, wherein the message comprises a simple object access protocol (SOAP) message.

22. (Previously Presented) An apparatus configured to identify a message source in a network, comprising:

means for receiving a method call from a client computer to invoke an object on a data server;

means for packaging the method call in a message to be sent from a client server to the data server via the network;

means, on the client server, for identifying, from an execution stack and through the use of a comparison algorithm, an object on the client computer that is invoking the object on the data server; and

means for transmitting the message to the data server.

EVIDENCE APPENDIX

Server: A server is a program that awaits and fulfills requests from client programs in the same or other computers. *See* Corba Glossary (http://www.ooportal.com/corba-fundamentals/glossary.html), last viewed March 10, 2008.

RELATED PROCEEDINGS APPENDIX

None.

IDL, Distributed Computing, MDA, UML, CWM

abstract algorithms
: Algorithms that do things without precise specifications of how or even what they are doing.

abstract class
: A class that contains one or more abstract methods, and therefore can never be instantiated. Abstract classes are defined so that other classes can extend them and make them concrete by implementing the abstract methods.

abstract method
: A method that has no implementation.

Abstract Window Toolkit (AWT)
: A collection of Java-supplied GUI components. These components provide that subset of functionality which is common to all native platforms.

accessor method
: A method which allows other classes to read (but not change) the values of the attributes. Accessor methods are also known as "getter methods." Accessor methods are public methods that merely return the value of the attribute. See also getter method.

access protection
: Access protection allows programmers to restrict access to members of the class.

API
: Application Programming Interface. The API specifies the method used by a programmer when writing an application to access the behavior and state of classes and objects.

applet
: A program written in Java to run within a Java-enabled Web browser.

application
: Unlike an applet, a Java application does not require a Web browser to function. Applications are often referred to as "stand-alone."

argument
: A data item specified in a method call. An argument can be a literal value, a variable, or an expression.

array
: A collection of data items, all of the same type, in which the position of each item is uniquely designated by a discrete type.

ASCII
: American Standard Code for Information Interchange. A standard assignment of 7-bit numeric codes to characters.

attribute
: The attributes of the class are the values that define the state of the class. Attributes are also referred to as fields or variables.

Bean
: A reusable software component. Beans can be combined to create an application.

boolean
: Refers to an expression or variable that can have only a true or false value. Java provides the boolean type and the literal values true and false.

CASE tool
: A CASE tool creates design specifications for programmers and software designers. CASE stands for "Computer Assisted Software Engineering." A good CASE tool not only draws diagrams; it also can convert those diagrams to compilable source code in any of variety of languages.

children or child

A term occasionally substituted for subclass.

class
: In Java, a type that defines the implementation of a particular kind of object. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class.

client
: Any object or class outside the pattern; generally one that only knows about the public interface the pattern and its classes present, rather than about its private implementation.

comment
: In a program, the explanatory text that is ignored by the compiler. In Java programs, comments are delimited using // or /*...*/.

compiler
: A program to translate source code into code to be executed by a computer. The Java compiler translates Java source code into Java bytecode.

composition
: Creating objects with other objects as members. Composition should be used when a "has-a" relationship applies.

constructor
: A pseudo-method that creates an object.

data structure
: Describes how data is arranged in the structure.

delegation
: This pattern is one where a given object provides an interface to a set of operations. However, the actual work for those operations is performed by one or more other objects.

deprecation
: Refers to a class or method that is no longer important, and will probably cease to exist in the future.

early binding
: Early binding assumes that everything is known at compile time. It is always obvious which method will be invoked in which class at which point in the flow of a program. Early binding is slightly faster than late binding and is used exclusively by procedural languages like C and FORTRAN.

encapsulation
: The localization of knowledge within a module. Because objects encapsulate data and implementation, the user of an object can view the object as a black box that provides services. Instance variables and methods can be added, deleted, or changed, but as long as the services provided by the object remain the same, code that uses the object can continue to use it without being rewritten. See also instance variable, instance method.

exception
: An event during program execution that prevents the program from continuing normally; generally, an error.

extends
: Another term for "is a subclass of."

extrinsic
: Something other than the object itself keeps track of the state of the object.

field
: The fields of the class are the values that define the state of the class. Fields are also referred to as attributes or variables.

friendly access
: Providing friendly access to the fields of a class exposes the normally private implementation of a class to certain closely related classes that must access the class's fields frequently. Friendly members of a class are identified in a class diagram by underlining them.

getter method
: A method that allows other classes to read (but not change) the values of the attributes. Getter methods are also known as accessor methods. Getter methods are public methods that merely return the value to the attribute.

GUI
: Graphical User Interface. Refers to the techniques involved in using graphics, along with a keyboard and a mouse, to provide an easy-to-use interface.

hierarchy

A classification of relationships in which each item except the top one is a specialized form of the item above it. Each item can have one or more items below it in the hierarchy.

implementation
: The implementation of a pattern is a set of code that uses the pattern. It shows the details and tricks that are needed to make it work.

implementation inheritance
: Implementation inheritance allows a superclass and a subclass to share identical code for the same method. This sort of inheritance encourages code reuse.

inheritance
: The concept of classes automatically containing the variables and methods defined in their supertypes. See also superclass, subclass.

instance
: An object of a particular class.

interface
: The public members of a class.

interoperability
: With respect to software, the term interoperability is used to describe the capability of different programs to exchange data via a common set of business procedures, and to read and write the same file formats and use the same protocols.

Java
: An object-oriented programming language developed by Sun Microsystems.
A "write-once, run-anywhere" programming language.

JavaBeans
: A portable, platform-independent reusable component model.

Java Development Kit (JDK)
: A software development environment for writing applets and application in Java.

late binding
: Late binding allows design decisions to be deferred until runtime. Late binding is generally more flexible, and is used exclusively by Smalltalk and Java.

literal
: The basic representation of any integer, floating point, or character value. For example, 3.0 is a single-precision floating point literal, and "a" is a character literal.

member
: A field or method of a class. Unless specified otherwise, a member is not static.

method
: The operations of a class are stored as methods. A method, similar to a function in a procedural language like C, represents an action an object takes, or an operation the class performs.

method overloading
: Method overloading permits two methods (or constructors) to have the same name as each other, provided they can be distinguished by their argument lists.

multiple inheritance
: Multiple inheritance allows a single subclass to extend more than one superclass.

multithreaded
: Describes a program that is designed to have parts of its code execute concurrently.

mutator method
: Mutator methods allow nonsibling objects to change the state of an object. Typically, a mutator method is a public method. Mutator methods return as void and generally take a single argument of the same type they are setting. Mutator methods are also known as "setter methods." See also setter method.

noargs constructor
: A constructor that does not take arguments. They are sometimes referred to as "no-initializer constructors."

object
: The principal building block of object-oriented programs. Each object is a programming unit consisting of data and functionality.

object composition
: Other objects are stored as pointers or references inside the object that provides the interface to clients. Object composition is a powerful yet often overlooked tool in the OOP programmer's toolbox. Structural patterns show you how to take advantage of it.

object-oriented design
: A software design method that models the characteristics of abstract or real objects using classes and objects.

object-oriented programming
: A programming methodology that uses distinct units of functionality, with each unit called an object.

operations
: The operations of a class define what an object does.

operations compartment
: The operations compartment is the box at the bottom of the class diagram that contains method names.

Object Request Broker
: An Object Request Broker (ORB) is the programming that acts as a mediator between a client request for a service from a distributed object or component and the completion of that request.
The ORB is the middleware that establishes the client-server relationships between objects.

overloaded method
: An overloaded method has the same name but a different argument list as another method in the same class.

overloading
: Using one identifier to refer to multiple items in the same scope.

overriding
: Replacing a method that would otherwise be inherited from a superclass with a new method defined in the subclass is called overriding the method.

parent
: A term occasionally substituted for superclass.

protected
: By declaring that a member of a class is protected, you indicate that subclasses are allowed to access the member, though other nondescendant classes are not. Protected members are preceded by a "#" in the class diagram.

read-only properties
: Attributes with an accessor method but no mutator method can be gotten but not set. These are called read-only properties.

responsibilities
: The responsibilities of a class are defined as things the class knows, or the attributes of the class, plus the things the class does, or the operations of the class.

return values
: Methods can return information to whoever called them. This information is commonly called the method's return value, and like a method argument it has a type. Unlike a method argument, the return value does not have a name.

Reverse engineering
: The vice versa direction-source code to diagrams-is commonly referred to as reverse engineering.

scope
: A characteristic of an identifier that determines where the identifier can be used. Most identifiers in Java have either class or local scope. Instance and class variables and methods have class scope; they can be used outside the class and its subclasses only by prefixing them with an instance of the class or (for class variables and methods) with the class name. All other variables are declared within methods and have local scope; they can be used only within the enclosing block.

server
: A server is a program that awaits and fulfills requests from client programs in the same or other computers.

setter method
: Setter methods allow nonsibling objects to change the state of an object. Typically, a setter method is a public method. Setter methods return as void and generally take a single argument of the same type they are setting. Setter methods are also known as "mutator methods." See also mutator method.

sibling
: Objects of the same class are called siblings. Siblings are allowed to access the private elements of other siblings.

signature
: The signature of a method is its list of argument types.

static field
: Another name for "class variable."

static member
: A member that is shared by all variables of that class and is stored in only one place. Static members are identified by underlining them in a class diagram.

static method
: Another name for "class method."

subclass
: A class that is derived from a particular class, perhaps with one or more classes in between.

superclass
: A class from which a particular class is derived, perhaps with one or more classes in between.

try it
: Once you've read the definition, you can choose to view all the terms inthe glossary, or simply close the glossary window.

variable
: The variables of the class are the values that define the state of the class. Variables are also referred to as "attributes" or "fields."

void
: A method that does not return a value is considered void.

write-only properties
: Attributes with a mutator method but no accessor method can be set but not gotten. These are called write-only properties.